$\mathbf{si}_{u}nit_{p}andas$

Release 0.0.1

Dominic Davis-Foster

Apr 12, 2021

CONTENTS

Custom Pandas dtypes for values with SI units.

`si_unit_pandas` provides support for storing temperatures inside a pandas DataFrame using pandas' Extension Array Interface

Docs	
Tests	
Activity	
QA	
Other	

from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/si_unit_pandas@master --user
```

CHAPTER

ONE

KEY CONCEPTS

1.1 CelsiusType

This is a data type (like `numpy.dtype('int64')` or `pandas.api.types.CategoricalDtype()`. For the most part, you won't interact with `CelsiusType` directly. It will be the value of the `.dtype` attribute on your arrays.

1.2 Example

```
from si_unit_pandas import TemperatureArray
import pandas as pd

TemperatureArray([10, 20, 30, 40, 50])
```

1.2.1 Usage

This document describes how to use the methods and classes provided by `si_unit_pandas`.

We'll assume that the following imports have been performed.

```
import pandas as pd
from si_unit_pandas import TemperatureArray, to_temperature
```

Parsing

First, you'll need some temperature data. Much like pandas' `pandas.to_datetime()`, `si_unit_pandas` provides `to_temperature()` for converting sequences of anything to a specialized array, `TemperatureArray` in this case.

From Strings

`to_temperature()` can parse a sequence strings where each element represents a temperature.

```
to_temperature(['10', '20', '30', '40', '50'])
```

From Numbers

`to_temperature()` can also parse a sequence of numbers.

```
to_temperature([10, 20, 30.0, 40.5, 50])
```

Pandas Integration

`TemperatureArray` satisfies pandas extension array interface, which means that it can safely be stored inside pandas' Series and DataFrame.

```
values = to_temperature([10, 20, 30.0, 40.5, 50])

ser = pd.Series(values)

df = pd.DataFrame({"temperatures": values})
```

Most pandas methods that make sense should work.

1.2.2 API Reference

Table of Contents

Custom Pandas dtypes for values with SI units.

Classes:

Celsius(value)	`float` subclass representing a temperature in Celsius.
CelsiusType()	Numpy dtype representing a temperature in degrees Celsius.
Fahrenheit([value])	`float` subclass representing a temperature in Fahrenheit.
TemperatureArray(data[, dtype, copy])	Holder for Temperatures.

Functions:

to_temperature(values)	Convert values to a *TemperatureArray*.

class Celsius(*value*)

Bases: `UserFloat`

`float` subclass representing a temperature in Celsius.

Convert a string or number to a floating point number, if possible.

Methods:

__abs__()	Return `abs(self)`.
__add__(other)	Return `self + value`.
__bool__()	True if self != 0.
__ceil__()	Finds the least Integral >= self.
__complex__()	complex(self) == complex(float(self), 0)
__divmod__(other)	Return `divmod(self, value)`.
__eq__(other)	Return `self == other`.
__float__()	Return `float(self)`.
__floor__()	Finds the greatest Integral <= self.
__floordiv__(other)	Return `self // value`.
__ge__(other)	Return `self >= other`.
__gt__(other)	Return `self > other`.
__int__()	Return `int(self)`.
__le__(other)	Return `self <= other`.
__lt__(other)	Return `self < other`.
__mod__(other)	Return `self % value`.
__mul__(other)	Return `self * value`.
__ne__(other)	Return `self != other`.
__neg__()	Return `- self`.
__pos__()	Return `+ self`.
__pow__(other[, mod])	Return `pow(self, value, mod)`.
__radd__(other)	Return `value + self`.
__rdivmod__(other)	Return `divmod(value, self)`.
__repr__()	Return a string representation of the temperature.
__rfloordiv__(other)	Return `value // self`.
__rmod__(other)	Return `value % self`.
__rmul__(other)	Return `value * self`.
__round__([ndigits])	Rounds self to ndigits decimal places, defaulting to 0.
__rpow__(other[, mod])	Return `pow(value, self, mod)`.
__rsub__(other)	Return `value - self`.
__rtruediv__(other)	Return `value / self`.
__str__()	Return the temperature as a string.
__sub__(other)	Return `value - self`.
__truediv__(other)	Return `self / value`.
__trunc__()	trunc(self): Truncates self to an Integral.
as_integer_ratio()	**rtype** `Tuple[int, int]`
conjugate()	Conjugate is a no-op for Reals.

continues on next page

Table 3 – continued from previous page

fromhex(_UserFloat__s)	**rtype** ~_F
hex()	**rtype** str
is_integer()	**rtype** bool

Attributes:

__slots__	
_abc_impl	
imag	Real numbers have no imaginary component.
real	Real numbers are their real component.

__abs__()
Return abs(self).

Return type ~_F

__add__(*other*)
Return self + value.

Return type ~_F

__bool__()
True if self != 0. Called for bool(self).

Return type bool

__ceil__()
Finds the least Integral >= self.

__complex__()
complex(self) == complex(float(self), 0)

__divmod__(*other*)
Return divmod(self, value).

Return type Tuple[~_F, ~_F]

__eq__(*other*)
Return self == other.

Return type bool

__float__()
Return float(self).

Return type float

__floor__()
Finds the greatest Integral <= self.

__floordiv__(*other*)
Return self // value.

Return type ~_F

__ge__(*other*)
Return `self >= other`.

Return type `bool`

__gt__(*other*)
Return `self > other`.

Return type `bool`

__int__()
Return `int(self)`.

Return type `int`

__le__(*other*)
Return `self <= other`.

Return type `bool`

__lt__(*other*)
Return `self < other`.

Return type `bool`

__mod__(*other*)
Return `self % value`.

Return type `~_F`

__mul__(*other*)
Return `self * value`.

Return type `~_F`

__ne__(*other*)
Return `self != other`.

Return type `bool`

__neg__()
Return `- self`.

Return type `~_F`

__pos__()
Return `+ self`.

Return type `~_F`

__pow__(*other*, *mod=None*)
Return `pow(self, value, mod)`.

Return type `~_F`

__radd__(*other*)
Return `value + self`.

Return type `~_F`

__rdivmod__(*other*)
Return `divmod(value, self)`.

Return type `Tuple[~_F, ~_F]`

__repr__()
Return a string representation of the temperature.

Return type str

__rfloordiv__(*other*)
Return `value // self`.

Return type ~_F

__rmod__(*other*)
Return `value % self`.

Return type ~_F

__rmul__(*other*)
Return `value * self`.

Return type ~_F

__round__(*ndigits=None*)
Rounds self to ndigits decimal places, defaulting to 0.

If ndigits is omitted or None, returns an Integral, otherwise returns a Real. Rounds half toward even.

Return type Union[int, float]

__rpow__(*other*, *mod=None*)
Return `pow(value, self, mod)`.

Return type ~_F

__rsub__(*other*)
Return `value - self`.

Return type ~_F

__rtruediv__(*other*)
Return `value / self`.

Return type ~_F

__slots__ = ()

__str__()
Return the temperature as a string.

Return type str

__sub__(*other*)
Return `value - self`.

Return type ~_F

__truediv__(*other*)
Return `self / value`.

Return type ~_F

__trunc__()
trunc(self): Truncates self to an Integral.

Returns an Integral i such that:

- i>0 iff self>0;
- abs(i) <= abs(self);
- for any Integral j satisfying the first two conditions, abs(i) >= abs(j) [i.e. i has "maximal" abs among those].

i.e. “truncate towards 0”.

Return type int

_abc_impl = <_abc_data object>

as_integer_ratio()

Return type Tuple[int, int]

conjugate()
Conjugate is a no-op for Reals.

classmethod fromhex(*_UserFloat__s*)

Return type ~_F

hex()

Return type str

property imag
Real numbers have no imaginary component.

is_integer()

Return type bool

property real
Real numbers are their real component.

class CelsiusType
Bases: ExtensionDtype

Numpy dtype representing a temperature in degrees Celsius.

Methods:

__eq__(other)	Check whether ‘other’ is equal to self.
__ne__(other)	Return self!=value.
__str__()	Return str(self).
_get_common_dtype(dtypes)	Return the common dtype, if one exists.
construct_array_type()	Return the array type associated with this dtype.
construct_from_string(string)	Construct a *CelsiusType* from a string.
is_dtype(dtype)	Check if we match ‘dtype’.

Attributes:

_is_boolean	Whether this dtype should be considered boolean.
_is_numeric	Whether columns with this dtype should be considered numeric.
_metadata	
kind	
na_value	Default NA value to use for this type.
name	
names	Ordered list of field names, or None if there are no fields.

Classes:

_record_type	alias of `builtins.float`
type	alias of `TemperatureBase`

__eq__(*other*)
Check whether 'other' is equal to self.

By default, 'other' is considered equal if either

- it's a string matching 'self.name'.
- it's an instance of this type and all of the attributes in `self._metadata` are equal between *self* and *other*.

other : Any

bool

Return type bool

__ne__(*other*)
Return self!=value.

Return type bool

__str__()
Return str(self).

Return type str

_get_common_dtype(*dtypes*)
Return the common dtype, if one exists.

Used in *find_common_type* implementation. This is for example used to determine the resulting dtype in a concat operation.

If no common dtype exists, return None (which gives the other dtypes the chance to determine a common dtype). If all dtypes in the list return None, then the common dtype will be "object" dtype (this means it is never needed to return "object" dtype from this method itself).

dtypes [list of dtypes] The dtypes for which to determine a common dtype. This is a list of np.dtype or ExtensionDtype instances.

Common dtype (np.dtype or ExtensionDtype) or None

Return type Union[dtype, ExtensionDtype, None]

property _is_boolean
Whether this dtype should be considered boolean.

By default, ExtensionDtypes are assumed to be non-numeric. Setting this to True will affect the behavior of several places, e.g.

- is_bool
- boolean indexing

Return type bool

property _is_numeric
Whether columns with this dtype should be considered numeric.

By default ExtensionDtypes are assumed to be non-numeric. They'll be excluded from operations that exclude non-numeric columns, like (groupby) reductions, plotting, etc.

Return type bool

_metadata: Tuple[str, ...] = ()

_record_type
alias of builtins.float

classmethod construct_array_type()
Return the array type associated with this dtype.

type

Return type Type[TemperatureArray]

classmethod construct_from_string(*string*)
Construct a *CelsiusType* from a string.

Parameters string

classmethod is_dtype(*dtype*)
Check if we match 'dtype'.

dtype [object] The object to check.

bool

The default implementation is True if

1. cls.construct_from_string(dtype) is an instance of cls.
2. dtype is an object and is an instance of cls
3. dtype has a dtype attribute, and any of the above conditions is true for dtype.dtype.

Return type bool

kind: str = 'O'

property na_value
Default NA value to use for this type.

This is used in e.g. ExtensionArray.take. This should be the user-facing "boxed" version of the NA value, not the physical NA value for storage. e.g. for JSONArray, this is an empty dictionary.

Return type object

name: str = 'celsius'

property names
Ordered list of field names, or None if there are no fields.

This is for compatibility with NumPy arrays, and may be removed in the future.

Return type Optional[List[str]]

type
alias of TemperatureBase

class Fahrenheit(*value=0.0*)
Bases: UserFloat

float subclass representing a temperature in Fahrenheit.

Convert a string or number to a floating point number, if possible.

Methods:

__abs__()	Return `abs(self)`.
__add__(other)	Return `self + value`.
__bool__()	True if self != 0.
__ceil__()	Finds the least Integral >= self.
__complex__()	complex(self) == complex(float(self), 0)
__divmod__(other)	Return `divmod(self, value)`.
__eq__(other)	Return `self == other`.
__float__()	Return `float(self)`.
__floor__()	Finds the greatest Integral <= self.
__floordiv__(other)	Return `self // value`.
__ge__(other)	Return `self >= other`.
__gt__(other)	Return `self > other`.
__int__()	Return `int(self)`.
__le__(other)	Return `self <= other`.
__lt__(other)	Return `self < other`.
__mod__(other)	Return `self % value`.
__mul__(other)	Return `self * value`.
__ne__(other)	Return `self != other`.
__neg__()	Return `- self`.
__pos__()	Return `+ self`.
__pow__(other[, mod])	Return `pow(self, value, mod)`.
__radd__(other)	Return `value + self`.
__rdivmod__(other)	Return `divmod(value, self)`.
__repr__()	Return a string representation of the temperature.
__rfloordiv__(other)	Return `value // self`.
__rmod__(other)	Return `value % self`.
__rmul__(other)	Return `value * self`.
__round__([ndigits])	Rounds self to ndigits decimal places, defaulting to 0.
__rpow__(other[, mod])	Return `pow(value, self, mod)`.
__rsub__(other)	Return `value - self`.
__rtruediv__(other)	Return `value / self`.
__str__()	Return the temperature as a string.
__sub__(other)	Return `value - self`.
__truediv__(other)	Return `self / value`.
__trunc__()	trunc(self): Truncates self to an Integral.
as_integer_ratio()	**rtype** `Tuple[int, int]`
conjugate()	Conjugate is a no-op for Reals.
fromhex(_UserFloat__s)	**rtype** `~_F`
hex()	**rtype** `str`
is_integer()	**rtype** `bool`

Attributes:

__slots__	
_abc_impl	
imag	Real numbers have no imaginary component.
real	Real numbers are their real component.

__abs__()
Return `abs(self)`.

Return type ~_F

__add__(*other*)
Return `self + value`.

Return type ~_F

__bool__()
True if self != 0. Called for bool(self).

Return type `bool`

__ceil__()
Finds the least Integral >= self.

__complex__()
complex(self) == complex(float(self), 0)

__divmod__(*other*)
Return `divmod(self, value)`.

Return type `Tuple`[~_F, ~_F]

__eq__(*other*)
Return `self == other`.

Return type `bool`

__float__()
Return `float(self)`.

Return type `float`

__floor__()
Finds the greatest Integral <= self.

__floordiv__(*other*)
Return `self // value`.

Return type ~_F

__ge__(*other*)
Return `self >= other`.

Return type `bool`

__gt__(*other*)
Return `self > other`.

Return type `bool`

__int__()
Return `int(self)`.

Return type `int`

__le__(*other*)
Return `self <= other`.

Return type `bool`

__lt__(*other*)
Return `self < other`.

Return type `bool`

__mod__(*other*)
Return `self % value`.

Return type `~_F`

__mul__(*other*)
Return `self * value`.

Return type `~_F`

__ne__(*other*)
Return `self != other`.

Return type `bool`

__neg__()
Return `- self`.

Return type `~_F`

__pos__()
Return `+ self`.

Return type `~_F`

__pow__(*other*, *mod=None*)
Return `pow(self, value, mod)`.

Return type `~_F`

__radd__(*other*)
Return `value + self`.

Return type `~_F`

__rdivmod__(*other*)
Return `divmod(value, self)`.

Return type `Tuple`[`~_F`, `~_F`]

__repr__()
Return a string representation of the temperature.

Return type `str`

__rfloordiv__(*other*)
Return `value // self`.

Return type `~_F`

__rmod__(*other*)
Return `value % self`.

Return type `~_F`

__rmul__(*other*)
Return `value * self`.

Return type ~_F

__round__(*ndigits=None*)
Rounds self to ndigits decimal places, defaulting to 0.

If ndigits is omitted or None, returns an Integral, otherwise returns a Real. Rounds half toward even.

Return type Union[int, float]

__rpow__(*other*, *mod=None*)
Return `pow(value, self, mod)`.

Return type ~_F

__rsub__(*other*)
Return `value - self`.

Return type ~_F

__rtruediv__(*other*)
Return `value / self`.

Return type ~_F

__slots__ = ()

__str__()
Return the temperature as a string.

Return type str

__sub__(*other*)
Return `value - self`.

Return type ~_F

__truediv__(*other*)
Return `self / value`.

Return type ~_F

__trunc__()
trunc(self): Truncates self to an Integral.

Returns an Integral i such that:

- i>0 iff self>0;
- abs(i) <= abs(self);
- for any Integral j satisfying the first two conditions, abs(i) >= abs(j) [i.e. i has "maximal" abs among those].

i.e. "truncate towards 0".

Return type int

_abc_impl = <_abc_data object>

as_integer_ratio()

Return type Tuple[int, int]

conjugate()
Conjugate is a no-op for Reals.

classmethod fromhex(*_UserFloat__s*)

Return type ~_F

hex()

Return type str

property imag
Real numbers have no imaginary component.

is_integer()

Return type bool

property real
Real numbers are their real component.

class TemperatureArray(*data*, *dtype=None*, *copy=False*)
Bases: BaseArray

Holder for Temperatures.

TemperatureArray is a container for Temperatures. It satisfies pandas' extension array interface, and so can be stored inside pandas.Series and pandas.DataFrame.

Attributes:

T	**rtype** ExtensionArray
__array_priority__	
_can_hold_na	
_dtype	
_itemsize	
_parser	
_typ	
can_hold_na	
dtype	The dtype for this extension array, *CelsiusType*.
na_value	The missing value.
nbytes	The number of bytes needed to store this object in memory.
ndim	
shape	Return a tuple of the array dimensions.
size	The number of elements in the array.

Methods:

__abs__()	
__add__(other)	
__and__(other)	
__contains__(item)	Return for *item in self*.
__delitem__(where)	
__divmod__(other)	
__eq__(other)	Return for *self == other* (element-wise equality).
__floordiv__(other)	
__ge__(other)	Return self>=value.
__getitem__(item)	Select a subset of self.

continues on next page

Table 11 – continued from previous page

__gt__(other)	Return self>value.
__iadd__(other)	
__iand__(other)	
__ifloordiv__(other)	
__ilshift__(other)	
__imatmul__(other)	
__imod__(other)	
__imul__(other)	
__invert__()	
__ior__(other)	
__ipow__(other)	
__irshift__(other)	
__isub__(other)	
__iter__()	Iterate over elements of the array.
__itruediv__(other)	
__ixor__(other)	
__le__(other)	Return self<=value.
__len__()	Returns the length of this array.
__lshift__(other)	
__lt__(other)	Return self<value.
__matmul__(other)	
__mod__(other)	
__mul__(other)	
__ne__(other)	Return for *self != other* (element-wise in-equality).
__neg__()	
__or__(other)	
__pos__()	
__pow__(other)	
__radd__(other)	
__rand__(other)	
__rdivmod__(other)	
__repr__()	Return repr(self).
__rfloordiv__(other)	
__rlshift__(other)	
__rmatmul__(other)	
__rmod__(other)	
__rmul__(other)	
__ror__(other)	
__rpow__(other)	
__rrshift__(other)	
__rshift__(other)	
__rsub__(other)	
__rtruediv__(other)	
__rxor__(other)	
__setitem__(key, value)	Set one or more values inplace.
__sub__(other)	
__truediv__(other)	
__xor__(other)	
_concat_same_type(to_concat)	Concatenate multiple arrays.
_format_values()	

continues on next page

Table 11 – continued from previous page

_formatter([boxed])	Formatting function for scalar values.
_formatting_values()	
_from_factorized(values, original)	Reconstruct an ExtensionArray after factorization.
_from_ndarray(data[, copy])	Zero-copy construction of a BaseArray from an ndarray.
_from_sequence(scalars[, dtype, copy])	Construct a new ExtensionArray from a sequence of scalars.
_from_sequence_of_strings(strings, *[, ...])	Construct a new ExtensionArray from a sequence of strings.
_isstringslice(where)	
_reduce(name, *[, skipna])	Return a scalar result of performing the reduction operation.
_values_for_argsort()	Return values for sorting.
_values_for_factorize()	Return an array and missing value suitable for factorization.
append(value)	Append a value to this TemperatureArray.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind])	Return the indices that would sort this array.
astype(dtype[, copy])	Returns the array with its values as the given dtype.
copy([deep])	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
isin(other)	Check whether elements of *self* are in *other*.
isna()	Indicator for whether each element is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
setitem(indexer, value)	Set the 'value' inplace.
shift([periods, fill_value])	Shift values by desired number.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
tolist()	Convert the array to a Python list.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
view([dtype])	Return a view on the array.

property T

Return type ExtensionArray

__abs__()

__add__(*other*)

__and__(*other*)

__array_priority__: int = 1000

__contains__(*item*)

Return for *item in self*.

Return type bool

__delitem__(*where*)

__divmod__(*other*)

__eq__(*other*)
Return for *self* == *other* (element-wise equality).

__floordiv__(*other*)

__ge__(*other*)
Return self>=value.

__getitem__(*item*)
Select a subset of self.

Parameters **item** (Union[int, slice, ndarray]) –

- int: The position in 'self' to get.
- slice: A slice object, where 'start', 'stop', and 'step' are integers or None.
- ndarray: A 1-d boolean NumPy ndarray the same length as 'self'

Return type scalar or ExtensionArray

Note: For scalar item, return a scalar value suitable for the array's type. This should be an instance of self.dtype.type.

For slice key, return an instance of ExtensionArray, even if the slice is length 0 or 1.

For a boolean mask, return an instance of ExtensionArray, filtered to the values where item is True.

__gt__(*other*)
Return self>value.

__iadd__(*other*)

__iand__(*other*)

__ifloordiv__(*other*)

__ilshift__(*other*)

__imatmul__(*other*)

__imod__(*other*)

__imul__(*other*)

__invert__()

__ior__(*other*)

__ipow__(*other*)

__irshift__(*other*)

__isub__(*other*)

__iter__()
Iterate over elements of the array.

__itruediv__(*other*)

__ixor__(*other*)

__le__(*other*)
Return self<=value.

__len__()
Returns the length of this array.

Return type int

__lshift__(*other*)

__lt__(*other*)
Return self<value.

__matmul__(*other*)

__mod__(*other*)

__mul__(*other*)

__ne__(*other*)
Return for *self != other* (element-wise in-equality).

__neg__()

__or__(*other*)

__pos__()

__pow__(*other*)

__radd__(*other*)

__rand__(*other*)

__rdivmod__(*other*)

__repr__()
Return repr(self).

Return type str

__rfloordiv__(*other*)

__rlshift__(*other*)

__rmatmul__(*other*)

__rmod__(*other*)

__rmul__(*other*)

__ror__(*other*)

__rpow__(*other*)

__rrshift__(*other*)

__rshift__(*other*)

__rsub__(*other*)

__rtruediv__(*other*)

__rxor__(*other*)

__setitem__(*key*, *value*)
Set one or more values inplace.

This method is not required to satisfy the pandas extension array interface.

key [int, ndarray, or slice] When called from, e.g. `Series.__setitem__`, `key` will be one of

- scalar int
- ndarray of integers.
- boolean ndarray
- slice object

value [ExtensionDtype.type, Sequence[ExtensionDtype.type], or object] value or values to be set of `key`.

None

__sub__(*other*)

__truediv__(*other*)

__xor__(*other*)

_can_hold_na = True

classmethod _concat_same_type(*to_concat*)
Concatenate multiple arrays.

Parameters **to_concat** (Sequence[ABCExtensionArray]) – sequence of this type

Return type ABCExtensionArray

_dtype: Type[pandas.core.dtypes.base.ExtensionDtype] = <si_unit_pandas.temperature.Ce

_format_values()

_formatter(*boxed=False*)
Formatting function for scalar values.

This is used in the default '__repr__'. The returned formatting function receives instances of your scalar type.

boxed [bool, default False] An indicated for whether or not your array is being printed within a Series, DataFrame, or Index (True), or just by itself (False). This may be useful if you want scalar values to appear differently within a Series versus on its own (e.g. quoted or not).

Callable[[Any], str] A callable that gets instances of the scalar type and returns a string. By default, `repr()` is used when `boxed=False` and `str()` is used when `boxed=True`.

Return type Callable[[Any], Optional[str]]

_formatting_values()

classmethod _from_factorized(*values*, *original*)
Reconstruct an ExtensionArray after factorization.

Parameters

- **values** (ndarray) – An integer ndarray with the factorized values.
- **original** (ExtensionArray) – The original ExtensionArray that factorize was called on.

See also:

`pandas.pandas.api.extensions.ExtensionArray.factorize()`

classmethod _from_ndarray(*data*, *copy=False*)
Zero-copy construction of a BaseArray from an ndarray.

Parameters

- **data** (ndarray) – This should have CelsiusType._record_type dtype
- **copy** (bool) – Whether to copy the data. Default False.

Return type ~_A

Returns

classmethod _from_sequence(*scalars*, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of scalars.

Parameters

- **scalars** (Iterable) – Each element will be an instance of the scalar type for this array, cls.dtype.type.
- **dtype** (*dtype, optional*) – Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray. Default None.
- **copy** (bool) – If True, copy the underlying data. Default False.

classmethod _from_sequence_of_strings(*strings*, ***, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of strings.

New in version 0.24.0.

strings [Sequence] Each element will be an instance of the scalar type for this array, cls.dtype.type.

dtype [dtype, optional] Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray.

copy [bool, default False] If True, copy the underlying data.

ExtensionArray

_isstringslice(*where*)

_itemsize: int = 16

property _parser

_reduce(*name*, ***, *skipna=True*, ***kwargs*)
Return a scalar result of performing the reduction operation.

name [str] Name of the function, supported values are: { any, all, min, max, sum, mean, median, prod, std, var, sem, kurt, skew }.

skipna [bool, default True] If True, skip NaN values.

****kwargs** Additional keyword arguments passed to the reduction function. Currently, *ddof* is the only supported kwarg.

scalar

TypeError : subclass does not define reductions

_typ = 'extension'

_values_for_argsort()
Return values for sorting.

ndarray The transformed values should maintain the ordering between values within the array.

ExtensionArray.argsort : Return the indices that would sort this array.

Return type ndarray

_values_for_factorize()
Return an array and missing value suitable for factorization.

values : ndarray

An array suitable for factorization. This should maintain order and be a supported dtype (Float64, Int64, UInt64, String, Object). By default, the extension array is cast to object dtype.

na_value [object] The value in *values* to consider missing. This will be treated as NA in the factorization routines, so it will be coded as *na_sentinel* and not included in *uniques*. By default, `np.nan` is used.

The values returned by this method are also used in `pandas.util.hash_pandas_object()`.

Return type `Tuple`[`ndarray`, `Any`]

append(*value*)
Append a value to this TemperatureArray.

Parameters **value** (`Union`[`float`, `str`, `Sequence`[`Union`[`float`, `str`]]])

argmax()
Return the index of maximum value.

In case of multiple occurrences of the maximum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmin

argmin()
Return the index of minimum value.

In case of multiple occurrences of the minimum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmax

argsort(*ascending=True*, *kind='quicksort'*, **args*, ***kwargs*)
Return the indices that would sort this array.

Parameters

- **ascending** (`bool`) – Whether the indices should result in an ascending or descending sort. Default `True`.
- **kind** (`Union`[`Literal`['quicksort'], `Literal`['mergesort'], `Literal`['heapsort']]) – {'quicksort', 'mergesort', 'heapsort'}, optional Sorting algorithm. Default `'quicksort'`.

*args and **kwargs are passed through to `numpy.argsort()`.

Return type `ndarray`

Returns Array of indices that sort `self`. If NaN values are contained, NaN values are placed at the end.

See also:

`numpy.argsort`: Sorting implementation used internally.

astype(*dtype*, *copy=True*)
Returns the array with its values as the given dtype.

Parameters

- **dtype**
- **copy** – If `True`, returns a copy of the array. Default `True`.

can_hold_na: bool = True

copy(*deep=False*)
Return a copy of the array.

Parameters **deep** (`bool`) – Default `False`.

Returns

Return type `ABCExtensionArray`

data
Type: `ndarray`

dropna()
Return ExtensionArray without NA values.

valid : ExtensionArray

property dtype
The dtype for this extension array, *CelsiusType*.

equals(*other*)
Return if another array is equivalent to this array.

Equivalent means that both arrays have the same shape and dtype, and all values compare equal. Missing values in the same location are considered equal (in contrast with normal equality).

other [ExtensionArray] Array to compare to this Array.

boolean Whether the arrays are equivalent.

Return type `bool`

factorize(*na_sentinel=- 1*)
Encode the extension array as an enumerated type.

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

---

factorize : Top-level factorize method that dispatches here.

`pandas.factorize()` offers a *sort* keyword as well.

Return type `Tuple`[`ndarray`, `ExtensionArray`]

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap.

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

ExtensionArray With NA/NaN filled.

isin(*other*)

Check whether elements of *self* are in *other*.

Comparison is done elementwise.

Parameters **other** (Union[float, str, Sequence[Union[float, str]]])

Return type ndarray

Returns A 1-D boolean ndarray with the same length as self.

isna()

Indicator for whether each element is missing.

property na_value

The missing value.

Example:

```
>>> BaseArray([]).na_value
numpy.nan
```

property nbytes

The number of bytes needed to store this object in memory.

Return type int

ndim: int = 1

ravel(*order='C'*)

Return a flattened view on this array.

order : {None, 'C', 'F', 'A', 'K'}, default 'C'

ExtensionArray

- Because ExtensionArrays are 1D-only, this is a no-op.
- The "order" argument is ignored, is for compatibility with NumPy.

Return type ExtensionArray

repeat(*repeats*, *axis=None*)

Repeat elements of a ExtensionArray.

Returns a new ExtensionArray where each element of the current ExtensionArray is repeated consecutively a given number of times.

repeats [int or array of ints] The number of repetitions for each element. This should be a non-negative integer. Repeating 0 times will return an empty ExtensionArray.

axis [None] Must be `None`. Has no effect but is accepted for compatibility with numpy.

repeated_array [ExtensionArray] Newly created ExtensionArray with repeated elements.

Series.repeat : Equivalent function for Series. Index.repeat : Equivalent function for Index. numpy.repeat : Similar method for `numpy.ndarray`. ExtensionArray.take : Take arbitrary positions.

```
>>> cat = pd.Categorical(['a', 'b', 'c'])
>>> cat
['a', 'b', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat(2)
['a', 'a', 'b', 'b', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat([1, 2, 3])
['a', 'b', 'b', 'c', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
```

searchsorted(*value*, *side='left'*, *sorter=None*)

Find indices where elements should be inserted to maintain order.

New in version 0.24.0.

Find the indices into a sorted array *self* (a) such that, if the corresponding elements in *value* were inserted before the indices, the order of *self* would be preserved.

Assuming that *self* is sorted:

side	returned index *i* satisfies
left	`self[i-1] < value <= self[i]`
right	`self[i-1] <= value < self[i]`

value [array_like] Values to insert into *self*.

side [{'left', 'right'}, optional] If 'left', the index of the first suitable location found is given. If 'right', return the last such index. If there is no suitable index, return either 0 or N (where N is the length of *self*).

sorter [1-D array_like, optional] Optional array of integer indices that sort array a into ascending order. They are typically the result of argsort.

array of ints Array of insertion points with the same shape as *value*.

numpy.searchsorted : Similar method from NumPy.

setitem(*indexer*, *value*)

Set the 'value' inplace.

property shape

Return a tuple of the array dimensions.

Return type `Tuple[int]`

shift(*periods=1*, *fill_value=None*)

Shift values by desired number.

Newly introduced missing values are filled with `self.dtype.na_value`.

New in version 0.24.0.

periods [int, default 1] The number of periods to shift. Negative values are allowed for shifting backwards.

fill_value [object, optional] The scalar value to use for newly introduced missing values. The default is `self.dtype.na_value`.

New in version 0.24.0.

ExtensionArray Shifted.

If `self` is empty or `periods` is 0, a copy of `self` is returned.

If `periods > len(self)`, then an array of size len(self) is returned, with all values filled with `self.dtype.na_value`.

Return type `ExtensionArray`

property size
The number of elements in the array.

Return type `int`

take(*indices*, *allow_fill=False*, *fill_value=None*)
Take elements from an array.

indices [sequence of int] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*.

- False: negative values in *indices* indicate positional indices from the right (the default). This is similar to `numpy.take()`.
- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used.

For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if necessary.

ExtensionArray

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

numpy.take : Take elements from an array along an axis. api.extensions.take : Take elements from an array.

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignment, with a *fill_value*.

Here's an example implementation, which relies on casting the extension array to object dtype. This uses the helper method `pandas.api.extensions.take()`.

```
def take(self, indices, allow_fill=False, fill_value=None):
    from pandas.core.algorithms import take

    # If the ExtensionArray is backed by an ndarray, then
    # just pass that here instead of coercing to object.
```

(continues on next page)

(continued from previous page)

```
    data = self.astype(object)

    if allow_fill and fill_value is None:
        fill_value = self.dtype.na_value

    # fill value should always be translated from the scalar
    # type for the array, to the physical storage type for
    # the data, before passing to take.

    result = take(data, indices, fill_value=fill_value,
                  allow_fill=allow_fill)
    return self._from_sequence(result, dtype=self.dtype)
```

to_numpy(*dtype=None*, *copy=False*, *na_value=<object object>*)

Convert to a NumPy ndarray.

New in version 1.0.0.

This is similar to `numpy.asarray()`, but may provide additional control over how the conversion is done.

dtype [str or numpy.dtype, optional] The dtype to pass to `numpy.asarray()`.

copy [bool, default False] Whether to ensure that the returned value is a not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

na_value [Any, optional] The value to use for missing values. The default value depends on *dtype* and the type of the array.

numpy.ndarray

Return type `ndarray`

tolist()

Convert the array to a Python list.

Return type `List`

transpose(**axes*)

Return a transposed view on this array.

Because ExtensionArrays are always 1D, this is a no-op. It is included for compatibility with np.ndarray.

Return type `ExtensionArray`

unique()

Compute the ExtensionArray of unique values.

uniques : ExtensionArray

Return type `ExtensionArray`

view(*dtype=None*)

Return a view on the array.

dtype [str, np.dtype, or ExtensionDtype, optional] Default None.

ExtensionArray or np.ndarray A view on the `ExtensionArray`'s data.

Return type `~ArrayLike`

to_temperature(*values*)

Convert values to a *TemperatureArray*.

Parameters **values** (Union[float, str, Sequence[Union[float, str]]])

Return type TemperatureArray

si_unit_pandas.__init__

Custom Pandas dtypes for values with SI units.

Classes:

Celsius(value)	float subclass representing a temperature in Celsius.
CelsiusType()	Numpy dtype representing a temperature in degrees Celsius.
Fahrenheit([value])	float subclass representing a temperature in Fahrenheit.
TemperatureArray(data[, dtype, copy])	Holder for Temperatures.

Functions:

to_temperature(values)	Convert values to a *TemperatureArray*.

class Celsius(*value*)

Bases: UserFloat

float subclass representing a temperature in Celsius.

Convert a string or number to a floating point number, if possible.

Methods:

__abs__()	Return abs(self).
__add__(other)	Return self + value.
__bool__()	True if self != 0.
__ceil__()	Finds the least Integral >= self.
__complex__()	complex(self) == complex(float(self), 0)
__divmod__(other)	Return divmod(self, value).
__eq__(other)	Return self == other.
__float__()	Return float(self).
__floor__()	Finds the greatest Integral <= self.
__floordiv__(other)	Return self // value.
__ge__(other)	Return self >= other.
__gt__(other)	Return self > other.
__int__()	Return int(self).
__le__(other)	Return self <= other.
__lt__(other)	Return self < other.
__mod__(other)	Return self % value.
__mul__(other)	Return self * value.
__ne__(other)	Return self != other.
__neg__()	Return - self.
__pos__()	Return + self.

continues on next page

Table 14 – continued from previous page

__pow__(other[, mod])	Return `pow(self, value, mod)`.
__radd__(other)	Return `value + self`.
__rdivmod__(other)	Return `divmod(value, self)`.
__repr__()	Return a string representation of the temperature.
__rfloordiv__(other)	Return `value // self`.
__rmod__(other)	Return `value % self`.
__rmul__(other)	Return `value * self`.
__round__([ndigits])	Rounds self to ndigits decimal places, defaulting to 0.
__rpow__(other[, mod])	Return `pow(value, self, mod)`.
__rsub__(other)	Return `value - self`.
__rtruediv__(other)	Return `value / self`.
__str__()	Return the temperature as a string.
__sub__(other)	Return `value - self`.
__truediv__(other)	Return `self / value`.
__trunc__()	trunc(self): Truncates self to an Integral.
as_integer_ratio()	**rtype** `Tuple[int, int]`
conjugate()	Conjugate is a no-op for Reals.
fromhex(_UserFloat__s)	**rtype** `~_F`
hex()	**rtype** `str`
is_integer()	**rtype** `bool`

Attributes:

__slots__	
_abc_impl	
imag	Real numbers have no imaginary component.
real	Real numbers are their real component.

__abs__()
Return `abs(self)`.

Return type `~_F`

__add__(*other*)
Return `self + value`.

Return type `~_F`

__bool__()
True if self != 0. Called for bool(self).

Return type `bool`

__ceil__()
Finds the least Integral >= self.

__complex__()
complex(self) == complex(float(self), 0)

__divmod__(*other*)
Return `divmod(self, value)`.

Return type `Tuple`[`~_F`, `~_F`]

__eq__(*other*)
Return `self == other`.

Return type `bool`

__float__()
Return `float(self)`.

Return type `float`

__floor__()
Finds the greatest Integral <= self.

__floordiv__(*other*)
Return `self // value`.

Return type `~_F`

__ge__(*other*)
Return `self >= other`.

Return type `bool`

__gt__(*other*)
Return `self > other`.

Return type `bool`

__int__()
Return `int(self)`.

Return type `int`

__le__(*other*)
Return `self <= other`.

Return type `bool`

__lt__(*other*)
Return `self < other`.

Return type `bool`

__mod__(*other*)
Return `self % value`.

Return type `~_F`

__mul__(*other*)
Return `self * value`.

Return type `~_F`

__ne__(*other*)
Return `self != other`.

Return type `bool`

__neg__()
Return `- self`.

Return type ~_F

__pos__()
Return `+ self`.

Return type ~_F

__pow__(*other*, *mod=None*)
Return `pow(self, value, mod)`.

Return type ~_F

__radd__(*other*)
Return `value + self`.

Return type ~_F

__rdivmod__(*other*)
Return `divmod(value, self)`.

Return type `Tuple`[~_F, ~_F]

__repr__()
Return a string representation of the temperature.

Return type `str`

__rfloordiv__(*other*)
Return `value // self`.

Return type ~_F

__rmod__(*other*)
Return `value % self`.

Return type ~_F

__rmul__(*other*)
Return `value * self`.

Return type ~_F

__round__(*ndigits=None*)
Rounds self to ndigits decimal places, defaulting to 0.

If ndigits is omitted or None, returns an Integral, otherwise returns a Real. Rounds half toward even.

Return type `Union`[`int`, `float`]

__rpow__(*other*, *mod=None*)
Return `pow(value, self, mod)`.

Return type ~_F

__rsub__(*other*)
Return `value - self`.

Return type ~_F

__rtruediv__(*other*)
Return `value / self`.

Return type ~_F

__slots__ = ()

__str__()
Return the temperature as a string.

Return type str

__sub__(*other*)
Return `value - self`.

Return type ~_F

__truediv__(*other*)
Return `self / value`.

Return type ~_F

__trunc__()
trunc(self): Truncates self to an Integral.

Returns an Integral i such that:

- i>0 iff self>0;
- abs(i) <= abs(self);
- for any Integral j satisfying the first two conditions, abs(i) >= abs(j) [i.e. i has "maximal" abs among those].

i.e. "truncate towards 0".

Return type int

_abc_impl = <_abc_data object>

as_integer_ratio()

Return type Tuple[int, int]

conjugate()
Conjugate is a no-op for Reals.

classmethod fromhex(*_UserFloat__s*)

Return type ~_F

hex()

Return type str

property imag
Real numbers have no imaginary component.

is_integer()

Return type bool

property real
Real numbers are their real component.

class CelsiusType
Bases: ExtensionDtype

Numpy dtype representing a temperature in degrees Celsius.

Methods:

__eq__(other)	Check whether 'other' is equal to self.
__ne__(other)	Return self!=value.
__str__()	Return str(self).
_get_common_dtype(dtypes)	Return the common dtype, if one exists.
construct_array_type()	Return the array type associated with this dtype.
construct_from_string(string)	Construct a *CelsiusType* from a string.
is_dtype(dtype)	Check if we match 'dtype'.

Attributes:

_is_boolean	Whether this dtype should be considered boolean.
_is_numeric	Whether columns with this dtype should be considered numeric.
_metadata	
kind	
na_value	Default NA value to use for this type.
name	
names	Ordered list of field names, or None if there are no fields.

Classes:

_record_type	alias of `builtins.float`
type	alias of `TemperatureBase`

__eq__(*other*)
Check whether 'other' is equal to self.

By default, 'other' is considered equal if either

- it's a string matching 'self.name'.
- it's an instance of this type and all of the attributes in `self._metadata` are equal between *self* and *other*.

other : Any

bool

Return type `bool`

__ne__(*other*)
Return self!=value.

Return type `bool`

__str__()
Return str(self).

Return type `str`

_get_common_dtype(*dtypes*)
Return the common dtype, if one exists.

Used in *find_common_type* implementation. This is for example used to determine the resulting dtype in a concat operation.

If no common dtype exists, return None (which gives the other dtypes the chance to determine a common dtype). If all dtypes in the list return None, then the common dtype will be "object" dtype (this means it is never needed to return "object" dtype from this method itself).

dtypes [list of dtypes] The dtypes for which to determine a common dtype. This is a list of np.dtype or ExtensionDtype instances.

Common dtype (np.dtype or ExtensionDtype) or None

Return type Union[dtype, ExtensionDtype, None]

property _is_boolean
Whether this dtype should be considered boolean.

By default, ExtensionDtypes are assumed to be non-numeric. Setting this to True will affect the behavior of several places, e.g.

- is_bool
- boolean indexing

Return type bool

property _is_numeric
Whether columns with this dtype should be considered numeric.

By default ExtensionDtypes are assumed to be non-numeric. They'll be excluded from operations that exclude non-numeric columns, like (groupby) reductions, plotting, etc.

Return type bool

_metadata: Tuple[str, ...] = ()

_record_type
alias of builtins.float

classmethod construct_array_type()
Return the array type associated with this dtype.

type

Return type Type[TemperatureArray]

classmethod construct_from_string(*string*)
Construct a *CelsiusType* from a string.

Parameters string

classmethod is_dtype(*dtype*)
Check if we match 'dtype'.

dtype [object] The object to check.

bool

The default implementation is True if

1. cls.construct_from_string(dtype) is an instance of cls.
2. dtype is an object and is an instance of cls
3. dtype has a dtype attribute, and any of the above conditions is true for dtype.dtype.

Return type bool

kind: `str = 'O'`

property na_value
Default NA value to use for this type.

This is used in e.g. ExtensionArray.take. This should be the user-facing "boxed" version of the NA value, not the physical NA value for storage. e.g. for JSONArray, this is an empty dictionary.

Return type `object`

name: `str = 'celsius'`

property names
Ordered list of field names, or None if there are no fields.

This is for compatibility with NumPy arrays, and may be removed in the future.

Return type `Optional[List[str]]`

type
alias of `TemperatureBase`

class Fahrenheit(*value=0.0*)
Bases: `UserFloat`

`float` subclass representing a temperature in Fahrenheit.

Convert a string or number to a floating point number, if possible.

Methods:

`__abs__`()	Return `abs(self)`.
`__add__`(other)	Return `self + value`.
`__bool__`()	True if self != 0.
`__ceil__`()	Finds the least Integral >= self.
`__complex__`()	complex(self) == complex(float(self), 0)
`__divmod__`(other)	Return `divmod(self, value)`.
`__eq__`(other)	Return `self == other`.
`__float__`()	Return `float(self)`.
`__floor__`()	Finds the greatest Integral <= self.
`__floordiv__`(other)	Return `self // value`.
`__ge__`(other)	Return `self >= other`.
`__gt__`(other)	Return `self > other`.
`__int__`()	Return `int(self)`.
`__le__`(other)	Return `self <= other`.
`__lt__`(other)	Return `self < other`.
`__mod__`(other)	Return `self % value`.
`__mul__`(other)	Return `self * value`.
`__ne__`(other)	Return `self != other`.
`__neg__`()	Return `- self`.
`__pos__`()	Return `+ self`.
`__pow__`(other[, mod])	Return `pow(self, value, mod)`.
`__radd__`(other)	Return `value + self`.
`__rdivmod__`(other)	Return `divmod(value, self)`.
`__repr__`()	Return a string representation of the temperature.
`__rfloordiv__`(other)	Return `value // self`.
`__rmod__`(other)	Return `value % self`.
`__rmul__`(other)	Return `value * self`.

continues on next page

Table 19 – continued from previous page

__round__([ndigits])	Rounds self to ndigits decimal places, defaulting to 0.
__rpow__(other[, mod])	Return `pow(value, self, mod)`.
__rsub__(other)	Return `value - self`.
__rtruediv__(other)	Return `value / self`.
__str__()	Return the temperature as a string.
__sub__(other)	Return `value - self`.
__truediv__(other)	Return `self / value`.
__trunc__()	trunc(self): Truncates self to an Integral.
as_integer_ratio()	**rtype** `Tuple`[`int`, `int`]
conjugate()	Conjugate is a no-op for Reals.
fromhex(_UserFloat__s)	**rtype** `~_F`
hex()	**rtype** `str`
is_integer()	**rtype** `bool`

Attributes:

__slots__	
_abc_impl	
imag	Real numbers have no imaginary component.
real	Real numbers are their real component.

__abs__()
Return `abs(self)`.

Return type `~_F`

__add__(*other*)
Return `self + value`.

Return type `~_F`

__bool__()
True if self != 0. Called for bool(self).

Return type `bool`

__ceil__()
Finds the least Integral >= self.

__complex__()
complex(self) == complex(float(self), 0)

__divmod__(*other*)
Return `divmod(self, value)`.

Return type `Tuple`[`~_F`, `~_F`]

__eq__(*other*)
Return `self == other`.

Return type `bool`

__float__()
Return `float(self)`.

Return type `float`

__floor__()
Finds the greatest Integral <= self.

__floordiv__(*other*)
Return `self // value`.

Return type `~_F`

__ge__(*other*)
Return `self >= other`.

Return type `bool`

__gt__(*other*)
Return `self > other`.

Return type `bool`

__int__()
Return `int(self)`.

Return type `int`

__le__(*other*)
Return `self <= other`.

Return type `bool`

__lt__(*other*)
Return `self < other`.

Return type `bool`

__mod__(*other*)
Return `self % value`.

Return type `~_F`

__mul__(*other*)
Return `self * value`.

Return type `~_F`

__ne__(*other*)
Return `self != other`.

Return type `bool`

__neg__()
Return `- self`.

Return type `~_F`

__pos__()
Return `+ self`.

Return type `~_F`

__pow__(*other*, *mod=None*)
Return `pow(self, value, mod)`.

Return type ~_F

__radd__(*other*)
Return `value + self`.

Return type ~_F

__rdivmod__(*other*)
Return `divmod(value, self)`.

Return type `Tuple`[~_F, ~_F]

__repr__()
Return a string representation of the temperature.

Return type `str`

__rfloordiv__(*other*)
Return `value // self`.

Return type ~_F

__rmod__(*other*)
Return `value % self`.

Return type ~_F

__rmul__(*other*)
Return `value * self`.

Return type ~_F

__round__(*ndigits=None*)
Rounds self to ndigits decimal places, defaulting to 0.

If ndigits is omitted or None, returns an Integral, otherwise returns a Real. Rounds half toward even.

Return type `Union`[`int`, `float`]

__rpow__(*other*, *mod=None*)
Return `pow(value, self, mod)`.

Return type ~_F

__rsub__(*other*)
Return `value - self`.

Return type ~_F

__rtruediv__(*other*)
Return `value / self`.

Return type ~_F

__slots__ = ()

__str__()
Return the temperature as a string.

Return type `str`

__sub__(*other*)
Return `value - self`.

Return type ~_F

__truediv__(*other*)
Return `self / value`.

Return type ~_F

__trunc__()
trunc(self): Truncates self to an Integral.

Returns an Integral i such that:

- i>0 iff self>0;
- abs(i) <= abs(self);
- for any Integral j satisfying the first two conditions, abs(i) >= abs(j) [i.e. i has "maximal" abs among those].

i.e. "truncate towards 0".

Return type `int`

_abc_impl = <_abc_data object>

as_integer_ratio()

Return type `Tuple[int, int]`

conjugate()
Conjugate is a no-op for Reals.

classmethod fromhex(*_UserFloat__s*)

Return type ~_F

hex()

Return type `str`

property imag
Real numbers have no imaginary component.

is_integer()

Return type `bool`

property real
Real numbers are their real component.

class TemperatureArray(*data*, *dtype=None*, *copy=False*)
Bases: `BaseArray`

Holder for Temperatures.

TemperatureArray is a container for Temperatures. It satisfies pandas' extension array interface, and so can be stored inside `pandas.Series` and `pandas.DataFrame`.

Attributes:

T	**rtype** `ExtensionArray`
__array_priority__	
_can_hold_na	

continues on next page

Table 21 – continued from previous page

_dtype	
_itemsize	
_parser	
_typ	
can_hold_na	
dtype	The dtype for this extension array, *CelsiusType*.
na_value	The missing value.
nbytes	The number of bytes needed to store this object in memory.
ndim	
shape	Return a tuple of the array dimensions.
size	The number of elements in the array.

Methods:

__abs__()	
__add__(other)	
__and__(other)	
__contains__(item)	Return for *item in self*.
__delitem__(where)	
__divmod__(other)	
__eq__(other)	Return for *self == other* (element-wise equality).
__floordiv__(other)	
__ge__(other)	Return self>=value.
__getitem__(item)	Select a subset of self.
__gt__(other)	Return self>value.
__iadd__(other)	
__iand__(other)	
__ifloordiv__(other)	
__ilshift__(other)	
__imatmul__(other)	
__imod__(other)	
__imul__(other)	
__invert__()	
__ior__(other)	
__ipow__(other)	
__irshift__(other)	
__isub__(other)	
__iter__()	Iterate over elements of the array.
__itruediv__(other)	
__ixor__(other)	
__le__(other)	Return self<=value.
__len__()	Returns the length of this array.
__lshift__(other)	
__lt__(other)	Return self<value.
__matmul__(other)	
__mod__(other)	
__mul__(other)	
__ne__(other)	Return for *self != other* (element-wise in-equality).
__neg__()	

continues on next page

Table 22 – continued from previous page

`__or__`(other)	
`__pos__`()	
`__pow__`(other)	
`__radd__`(other)	
`__rand__`(other)	
`__rdivmod__`(other)	
`__repr__`()	Return repr(self).
`__rfloordiv__`(other)	
`__rlshift__`(other)	
`__rmatmul__`(other)	
`__rmod__`(other)	
`__rmul__`(other)	
`__ror__`(other)	
`__rpow__`(other)	
`__rrshift__`(other)	
`__rshift__`(other)	
`__rsub__`(other)	
`__rtruediv__`(other)	
`__rxor__`(other)	
`__setitem__`(key, value)	Set one or more values inplace.
`__sub__`(other)	
`__truediv__`(other)	
`__xor__`(other)	
`_concat_same_type`(to_concat)	Concatenate multiple arrays.
`_format_values`()	
`_formatter`([boxed])	Formatting function for scalar values.
`_formatting_values`()	
`_from_factorized`(values, original)	Reconstruct an ExtensionArray after factorization.
`_from_ndarray`(data[, copy])	Zero-copy construction of a BaseArray from an ndarray.
`_from_sequence`(scalars[, dtype, copy])	Construct a new ExtensionArray from a sequence of scalars.
`_from_sequence_of_strings`(strings, *[, ...])	Construct a new ExtensionArray from a sequence of strings.
`_isstringslice`(where)	
`_reduce`(name, *[, skipna])	Return a scalar result of performing the reduction operation.
`_values_for_argsort`()	Return values for sorting.
`_values_for_factorize`()	Return an array and missing value suitable for factorization.
`append`(value)	Append a value to this TemperatureArray.
`argmax`()	Return the index of maximum value.
`argmin`()	Return the index of minimum value.
`argsort`([ascending, kind])	Return the indices that would sort this array.
`astype`(dtype[, copy])	Returns the array with its values as the given dtype.
`copy`([deep])	Return a copy of the array.
`dropna`()	Return ExtensionArray without NA values.
`equals`(other)	Return if another array is equivalent to this array.
`factorize`([na_sentinel])	Encode the extension array as an enumerated type.
`fillna`([value, method, limit])	Fill NA/NaN values using the specified method.

continues on next page

Table 22 – continued from previous page

isin(other)	Check whether elements of *self* are in *other*.
isna()	Indicator for whether each element is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
setitem(indexer, value)	Set the 'value' inplace.
shift([periods, fill_value])	Shift values by desired number.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
tolist()	Convert the array to a Python list.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
view([dtype])	Return a view on the array.

property T

Return type ExtensionArray

__abs__()

__add__(*other*)

__and__(*other*)

__array_priority__: int = 1000

__contains__(*item*)
Return for *item in self*.

Return type bool

__delitem__(*where*)

__divmod__(*other*)

__eq__(*other*)
Return for *self* == *other* (element-wise equality).

__floordiv__(*other*)

__ge__(*other*)
Return self>=value.

__getitem__(*item*)
Select a subset of self.

Parameters **item** (Union[int, slice, ndarray]) –

- int: The position in 'self' to get.
- slice: A slice object, where 'start', 'stop', and 'step' are integers or None.
- ndarray: A 1-d boolean NumPy ndarray the same length as 'self'

Return type scalar or ExtensionArray

Note: For scalar item, return a scalar value suitable for the array's type. This should be an instance of self.dtype.type.

For slice key, return an instance of ExtensionArray, even if the slice is length 0 or 1.

For a boolean mask, return an instance of `ExtensionArray`, filtered to the values where `item` is True.

__gt__(*other*)
Return self>value.

__iadd__(*other*)

__iand__(*other*)

__ifloordiv__(*other*)

__ilshift__(*other*)

__imatmul__(*other*)

__imod__(*other*)

__imul__(*other*)

__invert__()

__ior__(*other*)

__ipow__(*other*)

__irshift__(*other*)

__isub__(*other*)

__iter__()
Iterate over elements of the array.

__itruediv__(*other*)

__ixor__(*other*)

__le__(*other*)
Return self<=value.

__len__()
Returns the length of this array.

Return type `int`

__lshift__(*other*)

__lt__(*other*)
Return self<value.

__matmul__(*other*)

__mod__(*other*)

__mul__(*other*)

__ne__(*other*)
Return for *self != other* (element-wise in-equality).

__neg__()

__or__(*other*)

__pos__()

__pow__(*other*)

__radd__(*other*)

__rand__(*other*)

__rdivmod__(*other*)

__repr__()
Return repr(self).

Return type str

__rfloordiv__(*other*)

__rlshift__(*other*)

__rmatmul__(*other*)

__rmod__(*other*)

__rmul__(*other*)

__ror__(*other*)

__rpow__(*other*)

__rrshift__(*other*)

__rshift__(*other*)

__rsub__(*other*)

__rtruediv__(*other*)

__rxor__(*other*)

__setitem__(*key*, *value*)
Set one or more values inplace.

This method is not required to satisfy the pandas extension array interface.

key [int, ndarray, or slice] When called from, e.g. `Series.__setitem__`, `key` will be one of

- scalar int
- ndarray of integers.
- boolean ndarray
- slice object

value [ExtensionDtype.type, Sequence[ExtensionDtype.type], or object] value or values to be set of `key`.

None

__sub__(*other*)

__truediv__(*other*)

__xor__(*other*)

_can_hold_na = True

classmethod _concat_same_type(*to_concat*)
Concatenate multiple arrays.

Parameters **to_concat** (Sequence[ABCExtensionArray]) – sequence of this type

Return type ABCExtensionArray

_dtype: Type[pandas.core.dtypes.base.ExtensionDtype] = <si_unit_pandas.temperature.Ce

_format_values()

_formatter(*boxed=False*)
Formatting function for scalar values.

This is used in the default '__repr__'. The returned formatting function receives instances of your scalar type.

boxed [bool, default False] An indicated for whether or not your array is being printed within a Series, DataFrame, or Index (True), or just by itself (False). This may be useful if you want scalar values to appear differently within a Series versus on its own (e.g. quoted or not).

Callable[[Any], str] A callable that gets instances of the scalar type and returns a string. By default, `repr()` is used when `boxed=False` and `str()` is used when `boxed=True`.

Return type `Callable`[[`Any`], `Optional`[`str`]]

_formatting_values()

classmethod _from_factorized(*values*, *original*)
Reconstruct an ExtensionArray after factorization.

Parameters

- **values** (`ndarray`) – An integer ndarray with the factorized values.
- **original** (`ExtensionArray`) – The original ExtensionArray that factorize was called on.

See also:

`pandas.pandas.api.extensions.ExtensionArray.factorize()`

classmethod _from_ndarray(*data*, *copy=False*)
Zero-copy construction of a BaseArray from an ndarray.

Parameters

- **data** (`ndarray`) – This should have CelsiusType._record_type dtype
- **copy** (`bool`) – Whether to copy the data. Default `False`.

Return type `~_A`

Returns

classmethod _from_sequence(*scalars*, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of scalars.

Parameters

- **scalars** (`Iterable`) – Each element will be an instance of the scalar type for this array, `cls.dtype.type`.
- **dtype** (*dtype, optional*) – Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray. Default `None`.
- **copy** (`bool`) – If True, copy the underlying data. Default `False`.

classmethod _from_sequence_of_strings(*strings*, *, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of strings.

New in version 0.24.0.

strings [Sequence] Each element will be an instance of the scalar type for this array, `cls.dtype.type`.

dtype [dtype, optional] Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray.

copy [bool, default False] If True, copy the underlying data.

ExtensionArray

_isstringslice(*where*)

_itemsize: int = 16

property _parser

_reduce(*name*, *, *skipna=True*, ***kwargs*)

Return a scalar result of performing the reduction operation.

name [str] Name of the function, supported values are: { any, all, min, max, sum, mean, median, prod, std, var, sem, kurt, skew }.

skipna [bool, default True] If True, skip NaN values.

****kwargs** Additional keyword arguments passed to the reduction function. Currently, *ddof* is the only supported kwarg.

scalar

TypeError : subclass does not define reductions

_typ = 'extension'

_values_for_argsort()

Return values for sorting.

ndarray The transformed values should maintain the ordering between values within the array.

ExtensionArray.argsort : Return the indices that would sort this array.

Return type ndarray

_values_for_factorize()

Return an array and missing value suitable for factorization.

values : ndarray

An array suitable for factorization. This should maintain order and be a supported dtype (Float64, Int64, UInt64, String, Object). By default, the extension array is cast to object dtype.

na_value [object] The value in *values* to consider missing. This will be treated as NA in the factorization routines, so it will be coded as *na_sentinel* and not included in *uniques*. By default, `np.nan` is used.

The values returned by this method are also used in `pandas.util.hash_pandas_object()`.

Return type Tuple[ndarray, Any]

append(*value*)

Append a value to this TemperatureArray.

Parameters **value** (Union[float, str, Sequence[Union[float, str]]])

argmax()

Return the index of maximum value.

In case of multiple occurrences of the maximum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmin

argmin()
Return the index of minimum value.

In case of multiple occurrences of the minimum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmax

argsort(*ascending=True*, *kind='quicksort'*, **args*, ***kwargs*)
Return the indices that would sort this array.

Parameters

- **ascending** (bool) – Whether the indices should result in an ascending or descending sort. Default True.
- **kind** (Union[Literal['quicksort'], Literal['mergesort'], Literal['heapsort']]) – {'quicksort', 'mergesort', 'heapsort'}, optional Sorting algorithm. Default 'quicksort'.

*args and **kwargs are passed through to numpy.argsort().

Return type ndarray

Returns Array of indices that sort self. If NaN values are contained, NaN values are placed at the end.

See also:

numpy.argsort: Sorting implementation used internally.

astype(*dtype*, *copy=True*)
Returns the array with its values as the given dtype.

Parameters

- **dtype**
- **copy** – If True, returns a copy of the array. Default True.

can_hold_na: bool = True

copy(*deep=False*)
Return a copy of the array.

Parameters **deep** (bool) – Default False.

Returns

Return type ABCExtensionArray

data
Type: ndarray

dropna()
Return ExtensionArray without NA values.

valid : ExtensionArray

property dtype
The dtype for this extension array, *CelsiusType*.

equals(*other*)
Return if another array is equivalent to this array.

Equivalent means that both arrays have the same shape and dtype, and all values compare equal. Missing values in the same location are considered equal (in contrast with normal equality).

other [ExtensionArray] Array to compare to this Array.

boolean Whether the arrays are equivalent.

Return type bool

factorize(*na_sentinel=- 1*)
Encode the extension array as an enumerated type.

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

---

factorize : Top-level factorize method that dispatches here.

`pandas.factorize()` offers a *sort* keyword as well.

Return type Tuple[ndarray, ExtensionArray]

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap.

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

ExtensionArray With NA/NaN filled.

isin(*other*)
Check whether elements of *self* are in *other*.

Comparison is done elementwise.

Parameters **other** (Union[float, str, Sequence[Union[float, str]]])

Return type ndarray

Returns A 1-D boolean ndarray with the same length as self.

isna()
Indicator for whether each element is missing.

property na_value
The missing value.

Example:

```
>>> BaseArray([]).na_value
numpy.nan
```

property nbytes
The number of bytes needed to store this object in memory.

Return type int

ndim: int = 1

ravel(*order='C'*)
Return a flattened view on this array.

order : {None, 'C', 'F', 'A', 'K'}, default 'C'

ExtensionArray

- Because ExtensionArrays are 1D-only, this is a no-op.
- The "order" argument is ignored, is for compatibility with NumPy.

Return type ExtensionArray

repeat(*repeats*, *axis=None*)
Repeat elements of a ExtensionArray.

Returns a new ExtensionArray where each element of the current ExtensionArray is repeated consecutively a given number of times.

repeats [int or array of ints] The number of repetitions for each element. This should be a non-negative integer. Repeating 0 times will return an empty ExtensionArray.

axis [None] Must be None. Has no effect but is accepted for compatibility with numpy.

repeated_array [ExtensionArray] Newly created ExtensionArray with repeated elements.

Series.repeat : Equivalent function for Series. Index.repeat : Equivalent function for Index. numpy.repeat : Similar method for numpy.ndarray. ExtensionArray.take : Take arbitrary positions.

```
>>> cat = pd.Categorical(['a', 'b', 'c'])
>>> cat
['a', 'b', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat(2)
['a', 'a', 'b', 'b', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat([1, 2, 3])
['a', 'b', 'b', 'c', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
```

searchsorted(*value*, *side='left'*, *sorter=None*)
Find indices where elements should be inserted to maintain order.

New in version 0.24.0.

Find the indices into a sorted array *self* (a) such that, if the corresponding elements in *value* were inserted before the indices, the order of *self* would be preserved.

Assuming that *self* is sorted:

side	returned index *i* satisfies
left	`self[i-1] < value <= self[i]`
right	`self[i-1] <= value < self[i]`

value [array_like] Values to insert into *self*.

side [{'left', 'right'}, optional] If 'left', the index of the first suitable location found is given. If 'right', return the last such index. If there is no suitable index, return either 0 or N (where N is the length of *self*).

sorter [1-D array_like, optional] Optional array of integer indices that sort array a into ascending order. They are typically the result of argsort.

array of ints Array of insertion points with the same shape as *value*.

numpy.searchsorted : Similar method from NumPy.

setitem(*indexer*, *value*)
Set the 'value' inplace.

property shape
Return a tuple of the array dimensions.

Return type `Tuple[int]`

shift(*periods=1*, *fill_value=None*)
Shift values by desired number.

Newly introduced missing values are filled with `self.dtype.na_value`.

New in version 0.24.0.

periods [int, default 1] The number of periods to shift. Negative values are allowed for shifting backwards.

fill_value [object, optional] The scalar value to use for newly introduced missing values. The default is `self.dtype.na_value`.

New in version 0.24.0.

ExtensionArray Shifted.

If `self` is empty or `periods` is 0, a copy of `self` is returned.

If `periods > len(self)`, then an array of size len(self) is returned, with all values filled with `self.dtype.na_value`.

Return type `ExtensionArray`

property size
The number of elements in the array.

Return type `int`

take(*indices*, *allow_fill=False*, *fill_value=None*)
Take elements from an array.

indices [sequence of int] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*.

- False: negative values in *indices* indicate positional indices from the right (the default). This is similar to `numpy.take()`.
- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used.

For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if necessary.

ExtensionArray

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

numpy.take : Take elements from an array along an axis. api.extensions.take : Take elements from an array.

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignment, with a *fill_value*.

Here's an example implementation, which relies on casting the extension array to object dtype. This uses the helper method `pandas.api.extensions.take()`.

```
def take(self, indices, allow_fill=False, fill_value=None):
    from pandas.core.algorithms import take

    # If the ExtensionArray is backed by an ndarray, then
    # just pass that here instead of coercing to object.
    data = self.astype(object)

    if allow_fill and fill_value is None:
        fill_value = self.dtype.na_value

    # fill value should always be translated from the scalar
    # type for the array, to the physical storage type for
    # the data, before passing to take.

    result = take(data, indices, fill_value=fill_value,
                  allow_fill=allow_fill)
    return self._from_sequence(result, dtype=self.dtype)
```

to_numpy (*dtype=None*, *copy=False*, *na_value=<object object>*)

Convert to a NumPy ndarray.

New in version 1.0.0.

This is similar to `numpy.asarray()`, but may provide additional control over how the conversion is done.

dtype [str or numpy.dtype, optional] The dtype to pass to `numpy.asarray()`.

copy [bool, default False] Whether to ensure that the returned value is a not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

na_value [Any, optional] The value to use for missing values. The default value depends on *dtype* and the type of the array.

numpy.ndarray

Return type ndarray

tolist()
Convert the array to a Python list.

Return type List

transpose(**axes*)
Return a transposed view on this array.

Because ExtensionArrays are always 1D, this is a no-op. It is included for compatibility with np.ndarray.

Return type ExtensionArray

unique()
Compute the ExtensionArray of unique values.

uniques : ExtensionArray

Return type ExtensionArray

view(*dtype=None*)
Return a view on the array.

dtype [str, np.dtype, or ExtensionDtype, optional] Default None.

ExtensionArray or np.ndarray A view on the ExtensionArray's data.

Return type ~ArrayLike

to_temperature(*values*)
Convert values to a *TemperatureArray*.

Parameters **values** (Union[float, str, Sequence[Union[float, str]]])

Return type TemperatureArray

si_unit_pandas.base

Base functionality.

Classes:

NumPyBackedExtensionArrayMixin()	Mixin for pandas extension backed by a numpy array.

class NumPyBackedExtensionArrayMixin
Bases: ExtensionArray

Mixin for pandas extension backed by a numpy array.

Attributes:

T	**rtype** ExtensionArray
_can_hold_na	
_typ	

continues on next page

Table 24 – continued from previous page

dtype	The dtype for this extension array, *CelsiusType*.
nbytes	The number of bytes needed to store this object in memory.
ndim	Extension Arrays are only allowed to be 1-dimensional.
shape	Return a tuple of the array dimensions.
size	The number of elements in the array.

Methods:

__contains__(item)	Return for *item in self*.
__eq__(other)	Return for *self == other* (element-wise equality).
__getitem__(item)	Select a subset of self.
__iter__()	Iterate over elements of the array.
__len__()	Returns the length of this array.
__ne__(other)	Return for *self != other* (element-wise in-equality).
__repr__()	Return repr(self).
__setitem__(key, value)	Set one or more values inplace.
_concat_same_type(to_concat)	Concatenate multiple arrays.
_formatter([boxed])	Formatting function for scalar values.
_formatting_values()	
_from_factorized(values, original)	Reconstruct an ExtensionArray after factorization.
_from_sequence(scalars[, dtype, copy])	Construct a new ExtensionArray from a sequence of scalars.
_from_sequence_of_strings(strings, *[, ...])	Construct a new ExtensionArray from a sequence of strings.
_reduce(name, *[, skipna])	Return a scalar result of performing the reduction operation.
_values_for_argsort()	Return values for sorting.
_values_for_factorize()	Return an array and missing value suitable for factorization.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy([deep])	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
isna()	A 1-D array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
setitem(indexer, value)	Set the 'value' inplace.
shift([periods, fill_value])	Shift values by desired number.
take(indices, *[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
tolist()	Convert the array to a Python list.

continues on next page

Table 25 – continued from previous page

transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
view([dtype])	Return a view on the array.

property T

Return type ExtensionArray

__contains__(*item*)
Return for *item in self*.

Return type bool

__eq__(*other*)
Return for *self* == *other* (element-wise equality).

Return type ~ArrayLike

__getitem__(*item*)
Select a subset of self.

item [int, slice, or ndarray]

- int: The position in 'self' to get.
- slice: A slice object, where 'start', 'stop', and 'step' are integers or None
- ndarray: A 1-d boolean NumPy ndarray the same length as 'self'

item : scalar or ExtensionArray

For scalar item, return a scalar value suitable for the array's type. This should be an instance of self.dtype.type.

For slice key, return an instance of ExtensionArray, even if the slice is length 0 or 1.

For a boolean mask, return an instance of ExtensionArray, filtered to the values where item is True.

Return type Union[ExtensionArray, Any]

__iter__()
Iterate over elements of the array.

__len__()
Returns the length of this array.

Return type int

__ne__(*other*)
Return for *self* != *other* (element-wise in-equality).

Return type ~ArrayLike

__repr__()
Return repr(self).

Return type str

__setitem__(*key*, *value*)
Set one or more values inplace.

This method is not required to satisfy the pandas extension array interface.

key [int, ndarray, or slice] When called from, e.g. Series.__setitem__, key will be one of

- scalar int

- ndarray of integers.
- boolean ndarray
- slice object

value [ExtensionDtype.type, Sequence[ExtensionDtype.type], or object] value or values to be set of `key`.

None

_can_hold_na = True

classmethod _concat_same_type(*to_concat*)
Concatenate multiple arrays.

Parameters **to_concat** (Sequence[ABCExtensionArray]) – sequence of this type

Return type ABCExtensionArray

_dtype
Type: Type[ExtensionDtype]

_formatter(*boxed=False*)
Formatting function for scalar values.

This is used in the default '__repr__'. The returned formatting function receives instances of your scalar type.

boxed [bool, default False] An indicated for whether or not your array is being printed within a Series, DataFrame, or Index (True), or just by itself (False). This may be useful if you want scalar values to appear differently within a Series versus on its own (e.g. quoted or not).

Callable[[Any], str] A callable that gets instances of the scalar type and returns a string. By default, `repr()` is used when `boxed=False` and `str()` is used when `boxed=True`.

Return type Callable[[Any], Optional[str]]

_formatting_values()

classmethod _from_factorized(*values*, *original*)
Reconstruct an ExtensionArray after factorization.

Parameters

- **values** (ndarray) – An integer ndarray with the factorized values.
- **original** (ExtensionArray) – The original ExtensionArray that factorize was called on.

See also:

`pandas.pandas.api.extensions.ExtensionArray.factorize()`

classmethod _from_sequence(*scalars*, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of scalars.

Parameters

- **scalars** (Iterable) – Each element will be an instance of the scalar type for this array, `cls.dtype.type`.
- **dtype** (*dtype, optional*) – Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray. Default None.
- **copy** (bool) – If True, copy the underlying data. Default False.

classmethod _from_sequence_of_strings(*strings*, *, *dtype=None*, *copy=False*)
Construct a new ExtensionArray from a sequence of strings.

New in version 0.24.0.

strings [Sequence] Each element will be an instance of the scalar type for this array, `cls.dtype.type`.

dtype [dtype, optional] Construct for this particular dtype. This should be a Dtype compatible with the ExtensionArray.

copy [bool, default False] If True, copy the underlying data.

ExtensionArray

_reduce(*name*, *, *skipna=True*, ***kwargs*)
Return a scalar result of performing the reduction operation.

name [str] Name of the function, supported values are: { any, all, min, max, sum, mean, median, prod, std, var, sem, kurt, skew }.

skipna [bool, default True] If True, skip NaN values.

****kwargs** Additional keyword arguments passed to the reduction function. Currently, *ddof* is the only supported kwarg.

scalar

TypeError : subclass does not define reductions

_typ = 'extension'

_values_for_argsort()
Return values for sorting.

ndarray The transformed values should maintain the ordering between values within the array.

ExtensionArray.argsort : Return the indices that would sort this array.

Return type ndarray

_values_for_factorize()
Return an array and missing value suitable for factorization.

values : ndarray

An array suitable for factorization. This should maintain order and be a supported dtype (Float64, Int64, UInt64, String, Object). By default, the extension array is cast to object dtype.

na_value [object] The value in *values* to consider missing. This will be treated as NA in the factorization routines, so it will be coded as *na_sentinel* and not included in *uniques*. By default, `np.nan` is used.

The values returned by this method are also used in `pandas.util.hash_pandas_object()`.

Return type Tuple[ndarray, Any]

argmax()
Return the index of maximum value.

In case of multiple occurrences of the maximum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmin

argmin()
Return the index of minimum value.

In case of multiple occurrences of the minimum value, the index corresponding to the first occurrence is returned.

int

ExtensionArray.argmax

argsort(*ascending=True*, *kind='quicksort'*, **args*, ***kwargs*)
Return the indices that would sort this array.

Parameters

- **ascending** (bool) – Whether the indices should result in an ascending or descending sort. Default True.
- **kind** (Union[Literal['quicksort'], Literal['mergesort'], Literal['heapsort']]) – {'quicksort', 'mergesort', 'heapsort'}, optional Sorting algorithm. Default 'quicksort'.

*args and **kwargs are passed through to numpy.argsort().

Return type ndarray

Returns Array of indices that sort self. If NaN values are contained, NaN values are placed at the end.

See also:

numpy.argsort: Sorting implementation used internally.

astype(*dtype*, *copy=True*)
Cast to a NumPy array with 'dtype'.

dtype [str or dtype] Typecode or data-type to which the array is cast.

copy [bool, default True] Whether to copy the data, even if not necessary. If False, a copy is made only if the old dtype does not match the new dtype.

array [ndarray] NumPy ndarray with 'dtype' for its dtype.

copy(*deep=False*)
Return a copy of the array.

Parameters **deep** (bool) – Default False.

Returns

Return type ABCExtensionArray

dropna()
Return ExtensionArray without NA values.

valid : ExtensionArray

property dtype
The dtype for this extension array, *CelsiusType*.

equals(*other*)
Return if another array is equivalent to this array.

Equivalent means that both arrays have the same shape and dtype, and all values compare equal. Missing values in the same location are considered equal (in contrast with normal equality).

other [ExtensionArray] Array to compare to this Array.

boolean Whether the arrays are equivalent.

Return type bool

factorize(*na_sentinel=- 1*)
Encode the extension array as an enumerated type.

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

factorize : Top-level factorize method that dispatches here.

`pandas.factorize()` offers a *sort* keyword as well.

Return type Tuple[ndarray, ExtensionArray]

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap.

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

ExtensionArray With NA/NaN filled.

isna()
A 1-D array indicating if each value is missing.

na_values [Union[np.ndarray, ExtensionArray]] In most cases, this should return a NumPy ndarray. For exceptional cases like `SparseArray`, where returning an ndarray would be expensive, an ExtensionArray may be returned.

If returning an ExtensionArray, then

- `na_values._is_boolean` should be True
- *na_values* should implement `ExtensionArray._reduce()`
- `na_values.any` and `na_values.all` should be implemented

Return type ~ArrayLike

property nbytes
The number of bytes needed to store this object in memory.

Return type int

property ndim
Extension Arrays are only allowed to be 1-dimensional.

Return type int

ravel(*order='C'*)
Return a flattened view on this array.

order : {None, 'C', 'F', 'A', 'K'}, default 'C'

ExtensionArray

- Because ExtensionArrays are 1D-only, this is a no-op.
- The "order" argument is ignored, is for compatibility with NumPy.

Return type ExtensionArray

repeat(*repeats*, *axis=None*)
Repeat elements of a ExtensionArray.

Returns a new ExtensionArray where each element of the current ExtensionArray is repeated consecutively a given number of times.

repeats [int or array of ints] The number of repetitions for each element. This should be a non-negative integer. Repeating 0 times will return an empty ExtensionArray.

axis [None] Must be None. Has no effect but is accepted for compatibility with numpy.

repeated_array [ExtensionArray] Newly created ExtensionArray with repeated elements.

Series.repeat : Equivalent function for Series. Index.repeat : Equivalent function for Index. numpy.repeat : Similar method for numpy.ndarray. ExtensionArray.take : Take arbitrary positions.

```
>>> cat = pd.Categorical(['a', 'b', 'c'])
>>> cat
['a', 'b', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat(2)
['a', 'a', 'b', 'b', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
>>> cat.repeat([1, 2, 3])
['a', 'b', 'b', 'c', 'c', 'c']
Categories (3, object): ['a', 'b', 'c']
```

searchsorted(*value*, *side='left'*, *sorter=None*)
Find indices where elements should be inserted to maintain order.

New in version 0.24.0.

Find the indices into a sorted array *self* (a) such that, if the corresponding elements in *value* were inserted before the indices, the order of *self* would be preserved.

Assuming that *self* is sorted:

side	returned index *i* satisfies
left	`self[i-1] < value <= self[i]`
right	`self[i-1] <= value < self[i]`

value [array_like] Values to insert into *self*.

side [{'left', 'right'}, optional] If 'left', the index of the first suitable location found is given. If 'right', return the last such index. If there is no suitable index, return either 0 or N (where N is the length of *self*).

sorter [1-D array_like, optional] Optional array of integer indices that sort array a into ascending order. They are typically the result of argsort.

array of ints Array of insertion points with the same shape as *value*.

numpy.searchsorted : Similar method from NumPy.

setitem(*indexer*, *value*)
Set the 'value' inplace.

property shape
Return a tuple of the array dimensions.

Return type `Tuple[int]`

shift(*periods=1*, *fill_value=None*)
Shift values by desired number.

Newly introduced missing values are filled with `self.dtype.na_value`.

New in version 0.24.0.

periods [int, default 1] The number of periods to shift. Negative values are allowed for shifting backwards.

fill_value [object, optional] The scalar value to use for newly introduced missing values. The default is `self.dtype.na_value`.

New in version 0.24.0.

ExtensionArray Shifted.

If `self` is empty or `periods` is 0, a copy of `self` is returned.

If `periods > len(self)`, then an array of size len(self) is returned, with all values filled with `self.dtype.na_value`.

Return type `ExtensionArray`

property size
The number of elements in the array.

Return type `int`

take(*indices*, *, *allow_fill=False*, *fill_value=None*)
Take elements from an array.

indices [sequence of int] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*.

- False: negative values in *indices* indicate positional indices from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used.

For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if necessary.

ExtensionArray

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

numpy.take : Take elements from an array along an axis. api.extensions.take : Take elements from an array.

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignment, with a *fill_value*.

Here's an example implementation, which relies on casting the extension array to object dtype. This uses the helper method `pandas.api.extensions.take()`.

```
def take(self, indices, allow_fill=False, fill_value=None):
    from pandas.core.algorithms import take

    # If the ExtensionArray is backed by an ndarray, then
    # just pass that here instead of coercing to object.
    data = self.astype(object)

    if allow_fill and fill_value is None:
        fill_value = self.dtype.na_value

    # fill value should always be translated from the scalar
    # type for the array, to the physical storage type for
    # the data, before passing to take.

    result = take(data, indices, fill_value=fill_value,
                  allow_fill=allow_fill)
    return self._from_sequence(result, dtype=self.dtype)
```

Return type `ExtensionArray`

to_numpy (*dtype=None*, *copy=False*, *na_value=<object object>*)

Convert to a NumPy ndarray.

New in version 1.0.0.

This is similar to `numpy.asarray()`, but may provide additional control over how the conversion is done.

dtype [str or numpy.dtype, optional] The dtype to pass to `numpy.asarray()`.

copy [bool, default False] Whether to ensure that the returned value is a not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

na_value [Any, optional] The value to use for missing values. The default value depends on *dtype* and the type of the array.

numpy.ndarray

Return type ndarray

tolist()
Convert the array to a Python list.

Return type List

transpose(**axes*)
Return a transposed view on this array.

Because ExtensionArrays are always 1D, this is a no-op. It is included for compatibility with np.ndarray.

Return type ExtensionArray

unique()
Compute the ExtensionArray of unique values.

uniques : ExtensionArray

Return type ExtensionArray

view(*dtype=None*)
Return a view on the array.

dtype [str, np.dtype, or ExtensionDtype, optional] Default None.

ExtensionArray or np.ndarray A view on the ExtensionArray's data.

Return type ~ArrayLike

si_unit_pandas.parser

si_unit_pandas.temperature_array

1.2.3 Overview

si_unit_pandas uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install pre-commit with pip and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

1.2.4 Coding style

yapf-isort is used for code formatting.

It can be run manually via pre-commit:

```
$ pre-commit run yapf-isort -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

1.2.5 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6, run:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

1.2.6 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

1.2.7 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

1.2.8 Downloading source code

The `si_unit_pandas` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/si_unit_pandas

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/si_unit_pandas"
> Cloning into 'si_unit_pandas'...
> remote: Enumerating objects: 47, done.
> remote: Counting objects: 100% (47/47), done.
> remote: Compressing objects: 100% (41/41), done.
> remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
> Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
> Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

Building from source

The recommended way to build `si_unit_pandas` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

View the Function Index or browse the Source Code.

Browse the GitHub Repository

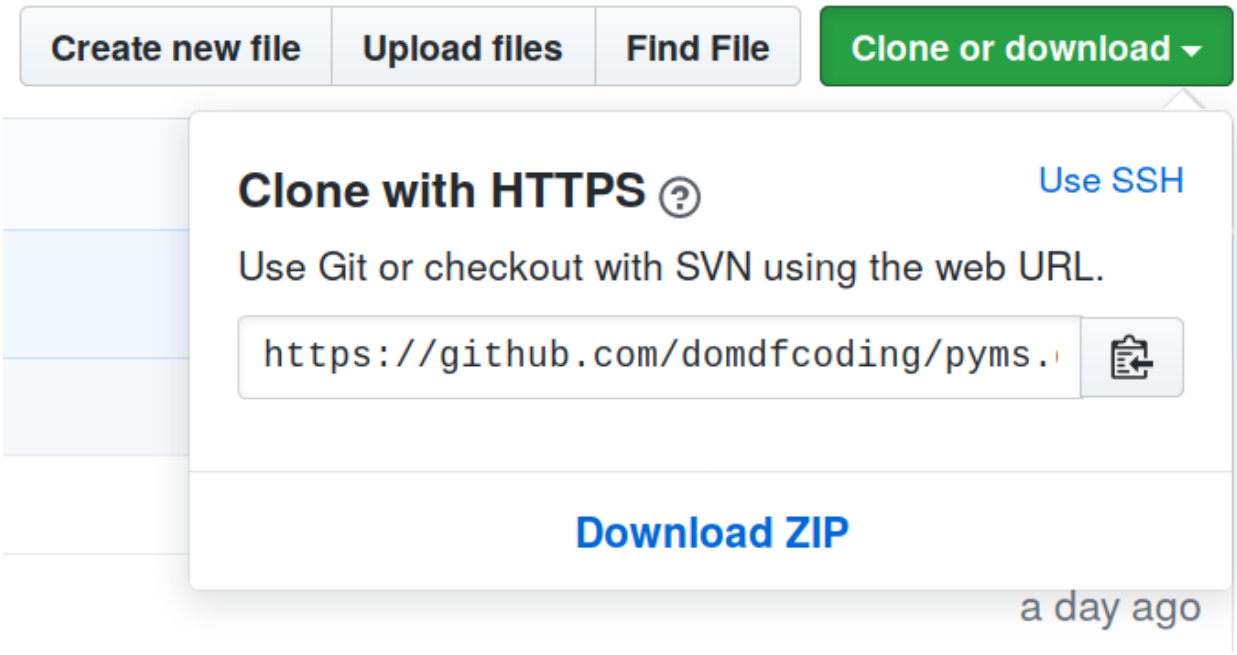

Fig. 1: Downloading a 'zip' file of the source code

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

H

I

K

M

N

P

R

S

T

U

V